An Extension of Interactive Scores for Multimedia Scenarios with Temporal Relations for Micro and Macro Controls

Mauricio Toro
LaBRI, Université de Bordeaux, France
mauriciotorob [at] gmail.com

Myriam Desainte-Catherine
LaBRI, IPB, U. de Bordeaux, France

Julien Castet
LaBRI, U. de Bordeaux, France

ABSTRACT

Software to design multimedia scenarios is usually based either on a fixed timeline or on cue lists, but both models are unrelated temporally. On the contrary, the formalism of interactive scores can describe multimedia scenarios with flexible and fixed temporal relations among the objects of the scenario, but cannot express neither temporal relations for micro controls nor signal processing. We extend interactive scores with such relations and with sound processing. We show some applications and we describe how they can be implemented in Pure Data. Our implementation has low average relative jitter even under high CPU load.

1. INTRODUCTION

Multimedia scenarios –such as interactive theater performances, interactive museum exhibitions and Electroacoustic music– are usually designed and controlled by computer programs. It is crucial that the software to execute such scenarios preserve the *macroform* and the *microform*. The macroform comprises the structure of the scenario (e.g., the tempo and the duration of the scenes, movements, parts and measures). The microform comprises the operations with samples (e.g., micro delays, articulation, and sound envelope). In this paper we deal with the macroform of multimedia content, but only with the microform of sound.

1.1 Problems

There are four problems with most existing multimedia scenario software: (1) time models are unrelated temporally, (2) they provide no hierarchy, (3) the different time scales are unrelated, and (4) schedulers are not appropriate for real-time. In what follows we explain each of them.

The first problem is that software to design multimedia scenarios is usually based either on a fixed timeline with a very precise script, such as *Pro Tools* [1] , or a more flexible script using cue lists, such as the theater cue manager *Qlab* [2] . Another software to design such scenarios is *Ableton Live* [3] . Live is often used in Electroacoustic music and performing arts because it allows to use both the fixed timeline and the cue lists, but the two time models are unrelated temporally. In fact, most software provide only one time model or they are unrelated temporally.

The second problem is that most software do not provide a hierarchy to represent the temporal objects of the scenario. As an example, using a hierarchy, it is possible to control the start or end of an object by controlling those from its parent. In interactive music, Vickery argues that a hierarchy is useful to control higher-order parameters of the piece; for instance, to control the volume dynamics, instead of the volume of each note [1].

The third problem is that the different time scales are often unrelated and cannot be controlled in the same tool. *Discrete user gestures* (e.g., clicking the mouse), *control events* (e.g., control messages) and *sound processing* have different sampling frequencies and computing models. As an example, the audio processing language *Csound* [4] has three types of variables with different sampling rates: instrument variables, control variables and audio variables.

As a consequence of having the time scales unrelated, it is difficult to associate, for instance, a human gesture to both control events and signal processing parameters in Csound. To control signal processing and control events by human gestures, *Max/*MSP and *Pure Data (Pd)* [2] are often used, but they do not provide an environment to design scenarios.

The fourth problem is that the most *soft real-time* schedulers, for instance those from Pd and Max, control both signals and control messages together and they do not support parallelism, thus they often fail to deliver control messages at the required time; for instance, when they work under high CPU load, which is common when they process video, 3D graphics and sound. We argue that in soft real-time, the usefulness of a result degrades after its deadline, thereby degrading the system's quality of service; whereas in *hard real-time* missing a deadline is a total system failure (e.g., flight control systems). We focus on soft real-time.

To solve the problem of scheduling and to write high-performance *digital signal processors (*DSPs*)* for Max and Pd, users often write C++ plugins to model loops and independent threads. C++ plugins solve part of the problem, but the control messages –for the input and output of these plugins– are handled by Max or Pd's schedulers.

Another solution for the scheduler problem –often used during live performance– is to open two or more instances of Max or Pd simultaneously, running different programs on each one. Nonetheless, synchronization is usually done

[1] http://www.avid.com/US/resources/digi-orientation
[2] http://figure53.com/qlab/
[3] http://www.ableton.com/
[4] http://www.csounds.com/

either manually during performance or by using *Open Sound Control (*OSC*)*, which adds more complexity and latency.

1.2 Practical and conceptual implications

The description of a multimedia scenario requires a consistent relationship between the representation of the scenario in the composition environment and the execution. Artistic creation requires a composition of events at different time scales. As an example, it is easy to describe that a video begins when the second string of a guitar arpeggio starts, but how can we achieve it in practice if the beginning of the notes of the arpeggio is controlled by the user?

The problem emerges at runtime. The example given above is very simple, but under high CPU load, a system interruption at the point of playing the arpeggio and the video can often lead to desynchronization. Usually, these eventualities are not considered by developers, as the quality of systems is evaluated according to an average performance. Nonetheless, during performance, it is desired that the system works well even under high CPU load.

The synchronization between the arpeggio and the video must be achieved in every execution. If it does not work for a performance, concert or show, the system performance is not satisfactory. Usually, artists prefer that an event is canceled if the event is not going to be properly synchronized with the other media. Users want a system that ensures that the events are either launched as they were defined in the score or they are not produced.

It is difficult to ensure determinism in the execution of multimedia processes (e.g., sound, video and 3D images). Some operating system like RT *Linux*[5] or *RedHawk*[6] include priority queues for processes to respect hard real-time constraints; however, in common operating systems, the user does not have this type of control.

This paper proposes a system to declare temporal constraints among multimedia processes that aims to ensure all temporal relations between events in the macroform and the microform of the scenario; however, our solution remains under the realm of soft real-time.

1.3 Interactive scores

There is a formalism to link both the fixed timeline and the cue list model. The formalism of *interactive scores* was proposed at the beginning of the century to describe scenarios with flexible and fixed temporal relations among temporal objects [3]. Examples of temporal objects are sounds, videos and light controls. The designer can specify that a video is played strictly before a light show, as an example of flexible temporal relations. The designer can also specify that a drum loop starts three seconds after the video, or between 10 and 15 seconds after, as an example of fixed temporal relations.

Interactive scores also include a hierarchy of temporal objects: An object contained inside another object must start after the execution of its father and must end before its father ends. In addition, by using fixed temporal relations on the first level of the hierarchy, it is possible to express absolute execution times for the events of the scenario.

A mathematical structural definition, an abstract semantics, formal properties of the scenarios, and an operational semantics of interactive scores was presented in [4].

The formalism of interactive scores has also encouraged the development of software. An implementation of interactive scores is *Virage*, which has been used for performing arts [5]. Another one is *Iscore* [6], used for composition of Electroacoustic music. Unfortunately, neither Virage nor iScore provide a satisfactory solution to control sound processing in real-time.

Virage can control different devices by the means of the OSC protocol and it can be used to model, for instance, curves that change the value of a DSP parameter for sound synthesis. Nonetheless, the values of these curves are sent at the control-event frequency and, therefore, its users cannot express temporal relations at the sound processing level; for instance, that one sound starts 500 μs after another.

1.4 Contributions

In this paper, we propose an extension to the interactive scores formalism to define DSPs for sound synthesis. This paper deals with the macrostructure of multimedia, but only microstructure of sound, and does not consider the structure of image, video or other media.

To define the microform of sound, we define a new type of temporal relations meant for high precision; for instance, to express micro delays. We also introduce dataflow relations; for instance, how the audio recorded by a temporal object is transferred to another object to filter it, add a micro delay, and then, send it to another temporal object to be diffused. The designer may define two views of the scenario: one for temporal relations and another one for dataflow (e.g., Fig. 10); otherwise, relations may overlap.

We also propose an encoding of the scenario into two models that interact during performance. (1) A model based on the *Non-deterministic Timed Concurrent Constraint* (`ntcc`) calculus [7] for concurrency, user interactions and temporal relations, and (2) a model based upon *Faust* [8] for sound processing and micro controls. The great advantage of having a formal model is that we could prove properties (e.g., playability) and predict the behavior of the system. In fact, the interoperability of ntcc and Faust has already been sketched in previous works [9, 10].

The novelty of our approach is using the constraints sent from `ntcc` to control Faust. We tested our applications in Pd, although they could also be compiled for Max or as a standalone program since both Faust and `ntcc` can be translated into C++ and Max. In fact, the final goal of our research is to develop a standalone program.

In what follows we briefly describe `ntcc`, Faust, and how they interact together.

1.5 Non-deterministic Timed Concurrent Constraint

In `ntcc`, a system is modeled in terms of processes adding to a common *store* partial information on the value of variables. Concurrent processes synchronize by blocking until a piece of information can be deduced from the store.

[5] http://www.windriver.com/index.html
[6] http://real-time.ccur.com/concurrent_redhawk_linux.aspx

Ntcc includes the notion of discrete time as a sequence of time units. Each time unit starts with a (possibly empty) store supplied by the environment. Processes scheduled for that time unit are then run until quiescence. The resulting store is the output at that time unit. Residual processes might also result. These are scheduled for the next (or any future) time unit and computation starts all over again. Ntcc has been used in many musical applications [11]. The semantics of ntcc processes are given in [7].

A model for interactive scores based upon ntcc is proposed in [4]. In this model, the store contains all the constraints from the temporal relations and the information of the events launched by the user. Temporal object processes synchronize themselves with the store.

Ntcc models can be simulated in a real-time setting using *Ntccrt* [12]. Ntccrt is based on *Gecode* [13]: state-of-the-art in constraint propagation. Ntccrt programs can be compiled into standalone programs, or plugins for Pd or Max. Users can use Pd to communicate any object with the Ntccrt plugin. In fact, Ntccrt can control all the available objects for audio processing defined in Pd, although our goal is to use Faust for such tasks.

1.6 Functional Audio STream (Faust)

Faust is a functional programming language for sound processing. In Faust, DSP algorithms are functions operating on signals. Faust programs are compiled into efficient C++ code that can be used in multiple programming languages and environments [14]. *Graphical user interface (*GUI*)* objects in Faust can be defined in the same way as other signals. We can control buttons, check boxes and integer inputs –originally designed for users– from Ntccrt (Fig. 1).

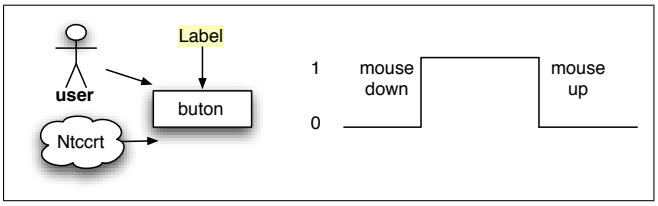

Figure 1. The signal delivered by the button reflects the user actions (or the Ntccrt output): one when the button is pressed; zero otherwise.

Although Faust programs can be compiled into efficient C++ programs, Faust programs are limited because all signals must have the same sampling rate. For that reason, Faust was recently extended for multirate [15]. With such an extension, Faust would be capable to handle signals at different frequencies. This is useful, for instance, for scenarios with different media such as audio and video. Unfortunately, this extension is not yet implemented, thus we only focus on sound processing.

Another extension of Faust is the *Pd-Faust* interface [14]. This interface is useful for DSPs that cannot be efficiently implemented in Pd because of a restriction of Pd: the 1-block minimum delay for feedback loops. An example of such a DSP is the *Karplus-Strong* algorithm [8]. Furthermore, Pd-Faust can also be used for other DSPs.

Finally, there is another reason to choose Faust: its extension for automatic parallelization and vectorization [16]. This extension has been proved to be very efficient; for instance, for the Karplus-Strong which we will use in several examples in this paper. Orlarey *et al.* found that using automatic parallelization, a program that simulates simultaneously 32 strings based on Karplus-Strong is twice faster using automatic parallelization [16].

1.7 Faust and ntcc interoperability

Ntcc can send constraints to Faust, but currently Faust cannot send information to ntcc because it requires subsampling. The constraints sent from ntcc cannot be partial information, such as $pitch > 3$ or $gain < 1$ because such information cannot be processed by Faust. Constraints must be equalities of the form $variable = constant$. Using Pure Data, we can communicate those values from ntcc to Faust by the means of number fields.

As an example, we present a possible interoperoperation between Faust and ntcc in Figure 2. On the one hand, ntcc can receive a user input each discrete time unit. If the value of the input is 1, ntcc communicates to Faust that the gain is 10; otherwise, if the user gives no input, ntcc communicates Faust that the gain is $1/10$. On the other hand, Faust takes an audio signal an multiplies by the gain value given by ntcc. In addition, Faust multiplies the signal by 2 if the current value of the audio input is less than 3.

Note that ntcc cannot take decisions based on the values of the audio signal because ntcc is not mean to handle audio signals, and Faust cannot take decision based on absence of information or partial information.

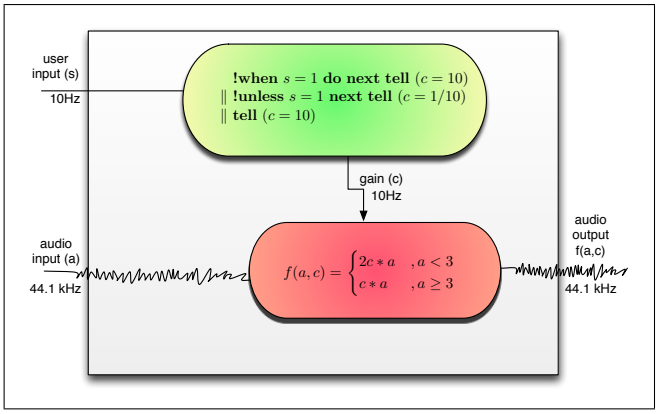

Figure 2. Example of ntcc and Faust interoperability.

1.8 Structure of the Paper

It is out of the scope of this paper to define formal semantics of interactive scores. Semantics of interactive scores were defined in [4]. It also out of the scope of this paper to fully describe the semantics of interactive scores and Faust interoperability. Such a semantics is to be defined in the *interactivity in the writing of time and interactions* (INEDIT) project supported by the *french research agency* (ANR). The purpose of this project is to explore the interoperability of interactive scores, Faust, and other

french computer music software. The project will start in fall 2012. This paper offers preliminary and encouraging results for INEDIT.

In what follows, we present the extension of interactive scores in Section 2; some applications developed with our framework in Section 3; quantitative results of the execution of the application in Section 4; and conclusions, results and future work in Section 5.

2. INTERACTIVE SCORES WITH MICRO AND MACRO CONTROLS

Scenarios in interactive scores are represented by *temporal objects*, *temporal relations for micro and macro controls*, *interactive objects* and *dataflow relations*.

2.1 Temporal Objects

Temporal objects can be triggered by interactive objects (usually launched by the user) and several temporal objects can be active simultaneously. The duration of a temporal object is given by an interval of natural numbers (which may include ∞). A temporal object may contain other temporal objects: this hierarchy allows us to control the start or end of a temporal object by controlling the start or end of its parent.

Objects that do not have children, may have a sound synthesis process. A process is a Faust program that is active during the execution of the object. These processes include at least two input signals: to control its start and end. During the execution of a score, only one instance of a temporal object can be active simultaneously because scores are linear and loops are not considered in this extension.

2.2 Temporal Relations

Temporal relations provide a partial order for the execution of the temporal objects; for instance, to express precedence between two objects. In interactive scores, it is also possible to specify a variety of relations among temporal objects such as global constraints and conditional branching.

In this paper, we take into account scenarios limited to hierarchical relations represented as a directed tree, *point-to-point temporal relations* without disjunction nor inequality ($\neq$), and quantitative temporal relations [17]. The first `ntcc` model proposed in [18] is based on Allen's relations; fortunately, point-to-point relations can express all Allen's relations without disjunction [19]. We proposed a `ntcc` model with point-to-point relations in [4]. In this paper, we extend such a model to control Faust from `ntcc`.

In the model in [4], the relations between the start or end of two temporal objects are labeled with an interval of integers that represents the possible duration between the two points. Using ∞ in such intervals, it is possible to represent the relations $<, >, \leq, \geq$ and $=$ with their usual interpretation over natural numbers.

In this paper, we also include *high-precision temporal relations*. This new type of temporal relations between sound objects are meant to have higher precision and they are controlled by Faust. Temporal relations for sound-processing micro controls are labeled by an integer n, where n represents, for instance, a number of samples or microseconds. Nonetheless, we can also use this relations to represent durations of seconds. We represent graphically such relations with dashed arrows.

2.3 Dataflow Relations

A dataflow relation between objects a and b means that the audio outputs of a are connected to the audio inputs of b. If a has more outputs than b inputs, they are merged; if a has less outputs than b, they are split. The control inputs of a Faust subprocess are connected automatically depending on the dataflow, and the micro and macro controls.

As an example, the reader may see the *dataflow view* of a scenario in Figure 3. In such a scenario, a sound is recorded by the acquisition object, then the stream is passed to a delay object, and then is passed to a filter that adds gain. Finally, the stream is passed to an object that sends two copies of the stream to the output. In what follow, we describe another example.

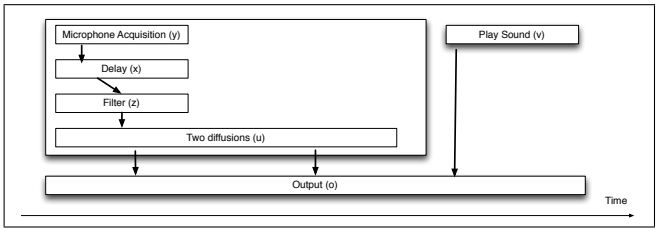

Figure 3. *Dataflow view* of a scenario. Thick arrows represent the flow of data through time.

2.4 Example: An Arpeggio with Three Strings

Karplus-Strong is an algorithm to generate metallic plucked-string sounds. It can be described in a few lines of Faust. In the Faust program presented by Orlarey *et al.* in [8], a button triggers the sound. We connect such button to a control signal sent from the Ntccrt plugin to the Faust plugin at the beginning of the temporal object. We also add another button to stop the sound of the string. In Pure Data (Pd), such buttons can be represented by *bang* or *toggle* objects that send messages to the plugin. In addition, we can use number fields as input for Faust. We use Pd for simplicity, but Pd is not required to integrate Ntccrt with Faust.

Figure 4 is a scenario that models an arpeggio of three strings using Karplus-Strong. The dataflow is simple: each audio outputs is merged into a single output. There are two types of temporal relations: some labeled with intervals in the order of seconds that will be handled by the *Ntccrt* plugin, and the high precision ones, in the order of samples, that will be handled by the Faust plugin.

The temporal constraints of the scenario are obtained from the duration of each temporal object, the hierarchy and from the temporal relations. For each temporal object, we add to the constraints: (1) "the start time of the object plus its duration is equal to the end time of the object" and (2) "the object starts after its father and ends before its father". For each temporal relation, we add the constraint "the time of the first point plus the duration in the relation is the time

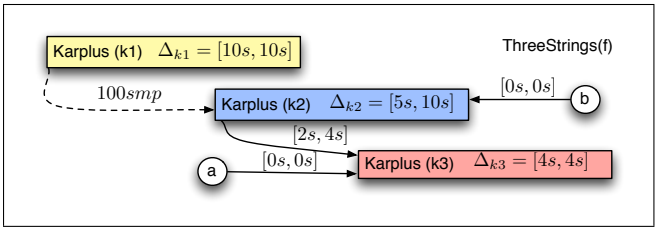

Figure 4. An example of a scenario. The durations in the temporal relations are labeled with seconds (s) and in the high precision temporal relations with samples (smp). Interactive objects are a and b.

of the second point". The temporal constraints of the score are explained in detail in [4].

Figure 5 is the constraint graph of the scenario in Figure 4. The `ntcc` model is parametric on the constraint graph, which can be obtained from the abstract semantics of the score, introduced in [4]. High precision relations are represented as zero durations in the constraint graph because they are controlled by Faust and not by Ntccrt, even if the duration in the relations were given in seconds.

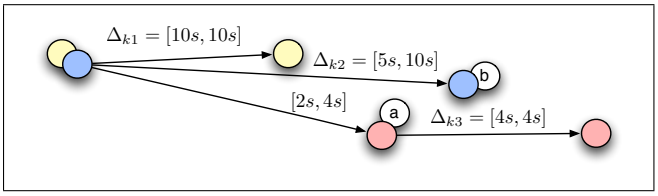

Figure 5. The temporal constraints of the scenario in Figure 4. The durations in the temporal relations are labeled with seconds (s) and the high precision temporal relations are considered as zero delays.

Figure 6 is the block diagram for the Faust program in charge of sound processing. Block diagram semantics are explained in detail in [8]. The inputs are controlled by the Ntccrt plugin. For simplicity, to avoid upsampling, control signals (e.g., e_{k1}, s_{k1} and e_{k2}) are replaced by Faust GUI buttons (Fig. 1). Interactive objects are represented by messages labeled by 1: If a message arrives, the interactive object must be launched. The audio output of each Karplus block is added together into a single output. Figure 7 is the Pure Data patch representing the scenario.

3. APPLICATIONS

We present some multimedia scenarios modeled in the extended formalism of interactive scores.

3.1 The Macro Structure of an Arpeggio Sequence

In Figure 8, we duplicate an arpeggio three times. The macroform is respected: The duration of each arpeggio is 10 seconds, but the start date and the durations of some notes can be controlled by the user with the freedom described in Figure 4. This problems shows how to solve the problem of having both time models (the cue list and the fixed timeline models) temporally related. In our framework, we can model the macroform of the arpeggio (e.g.,

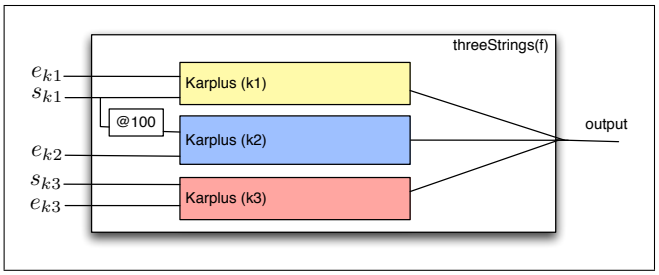

Figure 6. Block diagram representing the Faust process in charge of signal processing and the micro controls of the sound processors of the scenario. Signal processor @100 adds a delay of 100 samples to the signal s_{k1} (the start of the first string).

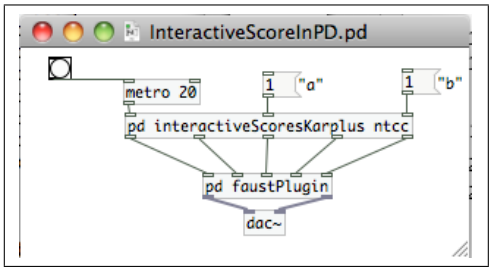

Figure 7. Pure Data patch representing the scenario in Figure 4. The Ntccrt plugin has only five outputs because the start of the second Karplus-Strong object ($k2$) is controlled directly from Faust. The internal clock of Ntccrt is controlled by a Pd *metronome* object with a period of 20 ms.

the duration of the notes and the global duration) and we can also model the microform (e.g., the microdelays handled by Faust and the delays among the notes that can be controlled by user interactions).

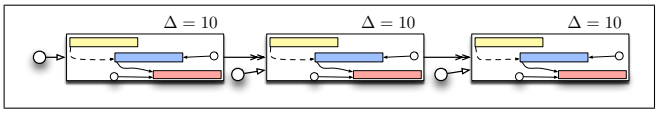

Figure 8. Three repetitions of a temporal object containing an arpeggio of three strings (described in Figure 4). The double-headed arrow represents an inequality ($\leq$) and a white-headed arrow represents an equality relation ($=$).

3.2 An Arpeggio without "Clicks"

There is a problem with the example in Figure 4: Interrupting abruptly the execution of the Karplus-Strong DSP causes perceptible "clicks". A solution to this problem is to gradually decrease the volume (or increase the attenuation parameter) before stopping the DSP. The value of 0.5 seconds is arbitrary, but it is fixed in the scenario, allowing us to know precisely the macroform of the scenario (e.g., its total duration). Therefore, instead of increasing the attenuation parameter indefinitely, we represent the attenuation with a temporal object, thus we can predict its duration and the global duration of the arpeggio.

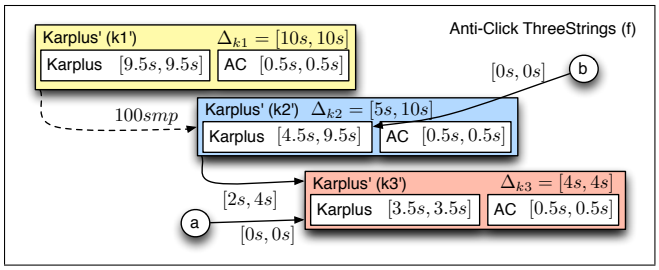

Figure 9. A modification of the scenario, presented in Figure 4, to remove "clicks". The Karplus objects simulate plucked-strings and the AC objects change the attenuation parameter of the strings gradually. The macroform of the scenario in Figure 4 is preserved intact.

3.3 Changing the Sound Source Perception

Small delays between the start of two temporal objects are usually not perceptible; however, in some cases –such as the example in Figure 10–, a small delay of 500 μs [7] between a sound played on the left channel and the same sound played on the right channel can change the way on which we perceive the sound source [8].

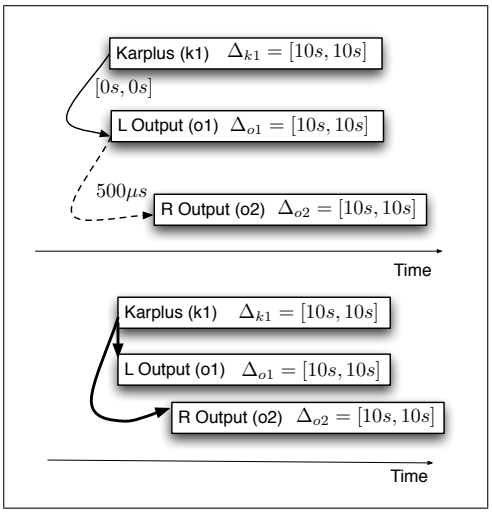

Figure 10. A scenario with a micro interval. First output is the left channel and second output is the right channel. First view is *temporal relations* and the second view is *dataflow relations*. It is better to represent separately both views of the score; otherwise, arrows will overlap.

4. RESULTS

We implemented the arpeggio of Figure 4. We tested three implementations of the Karplus-Strong in Pure Data (Pd): one from Colin Barry [9] that uses an instruction to define blocks of one sample (object *block~* 1), one from Johannes Kreidler [10] that uses one-sample delays (object *z~* 1), and one from Albert Gräf using a Faust plugin generated with Pd-Faust [11]. The interactive objects are launched automatically (at the latest possible time).

For each test, we played each arpeggio four times with a CPU load of 3% and four times with a load of 85%. We repeated each test ten times. The tests were performed in a 3.06 GHz Intel Core i3 processor on an iMac with a RAM memory of 4 Gb 1333 MHz DDR3, under Mac OS 10.6.8, using Pure Data extended 0.42 and Faust 0.9. To increase the CPU load, we ran several video processing operations from the *graphics environment for multimedia* (GEM) plugin for Pd. The CPU load values are approximatively and they were obtained using Mac OS X's *activity monitor*.

We calculated the average relative jitter of the micro- temporal structure of the scenario: the average time difference between the expected starting time of each string, with respect to the first string of the arpeggio, and the time obtained during execution. The average relative jitter using Faust is 500 μs with both a CPU load of 3% and 85%; on the contrary, the implementation from Colin Barry has a jitter of 7991 ms with a CPU load of 85% and the implementation from Johannes Kreidler has a jitter of 9231 ms with a CPU load of 85%. These values are very big and make the listening of the arpeggio incomprehensible. The average relative jitter was calculated using Matlab.

The Pd implementations of Karplus-Strong have also a limitation for high frequencies: They work well until 2000 Hz and Faust works well until 3000 Hz. Although this last result is the authors perception, we believe that the upper fundamental frequency limit may be due to the "chunk-sized" buffer delay in the feedback loop in Pd.

Another advantage of Faust is that the control signals in Faust can be delayed at sample level, whereas it is not possible to add sample delays to messages in Pd. In Pd, we need to delay the audio output instead of the control signals to produce such result. Finally, using Faust, sound processors could be automatically parallelized, improving its performance in many cases [16].

5. CONCLUSIONS

In this paper we extend the formalism of interactive scores with sound processing and micro controls for sound processors. We present an encoding of the scenario into a `ntcc` model –executed using the real-time capable interpreter *Ntccrt*– and a Faust program. Both programs interact during the performance of the scenario. We show how some interesting applications can be easily modeled in the formalism and how they can be executed in Pure Data (Pd).

Using Faust and Ntccrt, we achieved an efficient and real-time capable performance of a scenario –even under high CPU-load. Nonetheless, our final goal is to integrate Ntccrt and Faust in a standalone program. We argue that the solution we propose solves three of the problems we posed in the introduction.

First, time models are related temporally, for instance, we can specify that an object is executed strictly in the third second of execution, and we can can also express that another object is executed between two and five seconds

[7] This delay is equivalent to 22 samples at 44.1 kHz sampling rate.
[8] http://buschmeier.org/bh/study/soundperception/
[9] www.loomer.co.uk
[10] www.pd-tutorial.com
[11] http://docs.pure-lang.googlecode.com/hg/faust2pd.html

after the end of the previous object. Although in the execution the micro controls are managed by Faust and the macro controls by ntcc, it is also possible to express, for instance, that an object starts 500 microseconds after another, and it will end one second before another object.

Second, hierarchy is available in our model and it allows to constrain the execution times of the objects contained in another object.

Third, the system is appropriate, even under high CPU-load, to interact with a human in real-time, as shown in the quantitative results.

Unfortunately, different time scales are available in our tool, but they are temporally unrelated, as in many tools; for instance, is not possible to relate the frequency of the clock that controls ntcc discrete time units to the signal processing sampling rate.

Note that the score in Figure 4 is difficult to model in the existing tools presented in the introduction. Qlab and Live do not allow to model delays of 100 samples. Max and Csound allow to express delays of 100 samples, but it is very hard to synchronize processes whose durations are integer intervals such as $duration \in [5, 10]$.

The solution to these problems is relevant for the multimedia interaction domain because, in addition to sound processing, the computer may execute at the same time complex video and image operations. For that reason, we did the evaluation of our system under high CPU-load, obtained by executing several video processing operations concurrently.

5.1 Future Work

We believe that any Faust program could be translated into ntcc based on the results obtained by Rueda *et al.* in [9]. Rueda *et al.* translated the Karplus-Strong Faust program into ntcc. Although it is clear that the execution of a Ntccrt simulation cannot be done at sound processing sampling frequency, such translation could be used to verify properties of correctness of a scenario where ntcc and Faust interact (e.g., playability) as proposed in [9, 10].

We also propose to extend the implementation to handle audio files efficiently. *Libaudiostream* [12] is an audio library, developed at the french research institute *Grame* [13], to manipulate audio resources through the concept of streams using Faust programs.

Including Libaudiostream in our framework, it will be possible to design a scenario where a temporal object loads a sound file into memory, Faust filter it, and then, Faust plays the sound at the appropriate time. Precision is guaranteed because the time to load the file and process it is foreknown in the scenario. Currently, we have to rely on third-party programs, such as Pd, to do handle audio files, and to communicate the control signals from Ntccrt to Faust.

Acknowledgments

The authors are grateful to the reviewers for their comments. We also would like to thank the people from the Faust users list: Romain Michon, Albert Gräef and Julius Smith. We also like to thank Camilo Rueda and Yann Orlarey for their comments on our paper.

[12] http://libaudiostream.sourceforge.net/

[13] http://www.grame.fr/

6. REFERENCES

[1] L. Vickery, "Interactive control of higher order musical structures," in *Proc. of ACMC*. Victoria University, New Zealand: ACMA, July 2004.

[2] M. Puckette, T. Apel, and D. Zicarelli, "Real-time audio analysis tools for Pd and MSP," in *Proc. of ICMC '98*, Ann Arbor, USA, 1998. [Online]. Available: citeseer.ist.psu.edu/puckette98realtime.html

[3] A. Allombert, G. Assayag, and M. Desainte-Catherine, "A system of interactive scores based on petri nets," in *Proc. of SMC '07*, Athens, Greece, 2007.

[4] M. Toro, M. Desainte-Catherine, and C. Rueda, "Formal semantics for interactive music scores: A framework to design, specify properties and execute interactive scenarios," *Journal of Mathematics and Music. To appear in next volume*, 2012.

[5] A. Allombert, P. Baltazar, R. Marczak, M. Desainte-Catherine, and L. Garnier, "Designing an interactive intermedia sequencer from users requirements and theoretical background," in *Proc. of ICMC 2010*, 2010.

[6] A. Allombert, G. A. Assayag, and M. Desainte-Catherine, "Iscore: a system for writing interaction," in *Proc. of DIMEA '08*. New York, NY, USA: ACM, 2008, pp. 360–367.

[7] M. Nielsen, C. Palamidessi, and F. Valencia, "Temporal concurrent constraint programming: Denotation, logic and applications," *Nordic Journal of Computing*, vol. 1, no. 9, pp. 145–188, 2002. [Online]. Available: citeseer.ist.psu.edu/article/nielsen02temporal.html

[8] Y. Orlarey, D. Fober, and S. Letz, "Syntactical and semantical aspects of faust," *Soft Comput.*, vol. 8, no. 9, pp. 623–632, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1028287

[9] C. Rueda and F. Valencia, "A temporal concurrent constraint calculus as an audio processing framework," in *SMC '05*, 2005. [Online]. Available: http://www.lix.polytechnique.fr/~fvalenci/papers/ntcc-smc05.pdf

[10] M. Toro, "Structured musical interactive scores (short)," in *Proc. ICLP 2010*, 2010.

[11] C. Olarte, C. Rueda, G. Sarria, M. Toro, and F. Valencia, "Concurrent constraints models of music interaction," in *Constraint Programming in Music*, G. Assayag and C. Truchet, Eds. Hoboken, NJ, USA.: Wiley, 2011, ch. 6, pp. 133–153.

[12] M. Toro, C. Agón, G. Assayag, and C. Rueda, "Ntccrt: A concurrent constraint framework for real-time interaction," in *Proc. of ICMC '09*, Montreal, Canada, 2009.

[13] G. Tack, "Constraint propagation - models, techniques, implementation," Ph.D. dissertation, Saarland University, Germany, 2009.

[14] A. Gräf, "Interfacing pure data with faust," in *5th International Linux Audio Conference (LAC2007)*, LAC, Ed., 2007. [Online]. Available: http://www.grame.fr/pub/lac07.pdf

[15] P. Jouvelot and Y. Orlarey, "Dependent vector types for data structuring in multirate faust," *Computer Languages, Systems and Structures*, 2011.

[16] Y. Orlarey, D. Fober, and S. Letz, "Work stealing scheduler for automatic parallelization in faust," in *Proc. of Linux Audio Conference*, 2010.

[17] R. Gennari, "Temporal resoning and constraint programming - a survey," *CWI Quaterly*, vol. 11, pp. 3–163, 1998.

[18] A. Allombert, G. Assayag, M. Desainte-Catherine, and C. Rueda, "Concurrent constraint models for interactive scores," in *Proc. of SMC '06*, Marseille, France, 2006. [Online]. Available: http://www.labri.fr/publications/is/2006/AADR06

[19] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communication of ACM*, vol. 26, 1983.